Introduction to Machine Learning

Finding the Nearest Neighborhood

Andres Mendez-Vazquez

March 11, 2019

Outline

Outline

Geometric Data Structures

We want to have data structures for

Organization of points, lines, planes, ... to support faster processing

Geometric Data Structures

We want to have data structures for

Organization of points, lines, planes, ... to support faster processing

Applications

- Astrophysical simulation - Evolution of galaxies
- Graphics – computing object intersections
- Data compression
  - Points are representatives of 2x2 blocks in an image
  - Nearest neighbor search

Geometric Data Structures

We want to have data structures for

Organization of points, lines, planes, ... to support faster processing

Applications

- Astrophysical simulation - Evolution of galaxies
- Graphics – computing object intersections
- Data compression
  - Points are representatives of 2x2 blocks in an image
  - Nearest neighbor search

Geometric Data Structures

We want to have data structures for

Organization of points, lines, planes, ... to support faster processing

Applications

- Astrophysical simulation - Evolution of galaxies
- Graphics – computing object intersections
- Data compression
  - Points are representatives of 2x2 blocks in an image
  - Nearest neighbor search

Geometric Data Structures

We want to have data structures for

Organization of points, lines, planes, ... to support faster processing

Applications

- Astrophysical simulation - Evolution of galaxies
- Graphics – computing object intersections
- Data compression
  - Points are representatives of 2x2 blocks in an image
  - Nearest neighbor search

Geometric Data Structures

We want to have data structures for

Organization of points, lines, planes, ... to support faster processing

Applications

- Astrophysical simulation - Evolution of galaxies
- Graphics – computing object intersections
- Data compression
  - Points are representatives of 2x2 blocks in an image
  - Nearest neighbor search

Outline

KD Tree

Invented

By Jon Bentley, 1975

- Range queries
- Fast lookup

KD Tree

Invented

By Jon Bentley, 1975

Tree used to store spatial data.

- Nearest neighbor search.
- Range queries.
- Fast look-up.

KD Tree

Invented

By Jon Bentley, 1975

Tree used to store spatial data.

- Nearest neighbor search.
- Range queries.
- Fast look-up.

KD Tree

Invented

By Jon Bentley, 1975

Tree used to store spatial data.

- Nearest neighbor search.
- Range queries.
- Fast look-up.

Complexity

Nice

KD tree are guaranteed $\log_2 n$ depth where n is the number of points in the set.

Something Notable

Traditionally, KD trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

Complexity

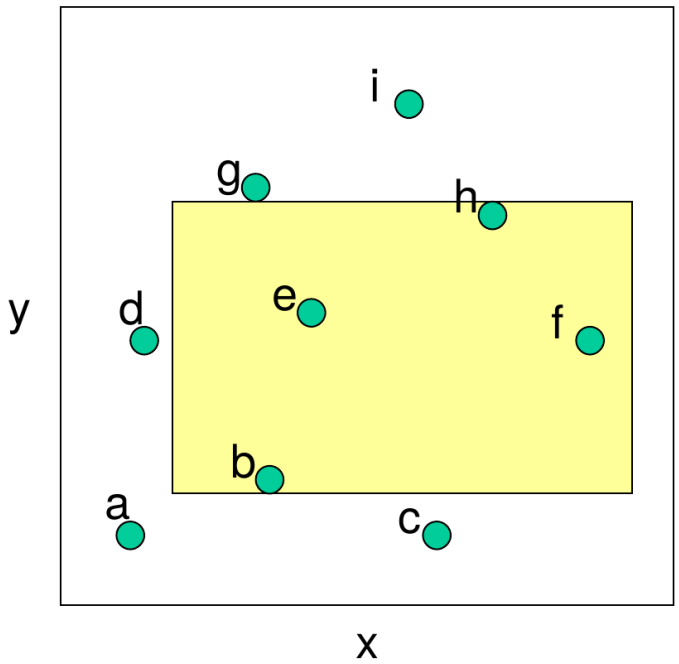
Nice

KD tree are guaranteed $\log_2 n$ depth where n is the number of points in the set.

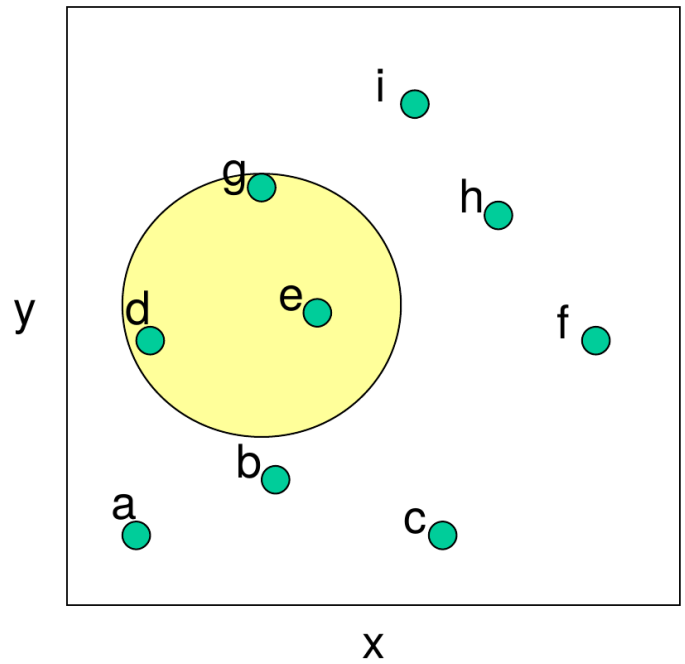
Something Notable

Traditionally, KD trees store points in d-dimensional space which are equivalent to vectors in d-dimensional space.

Example - Range Query

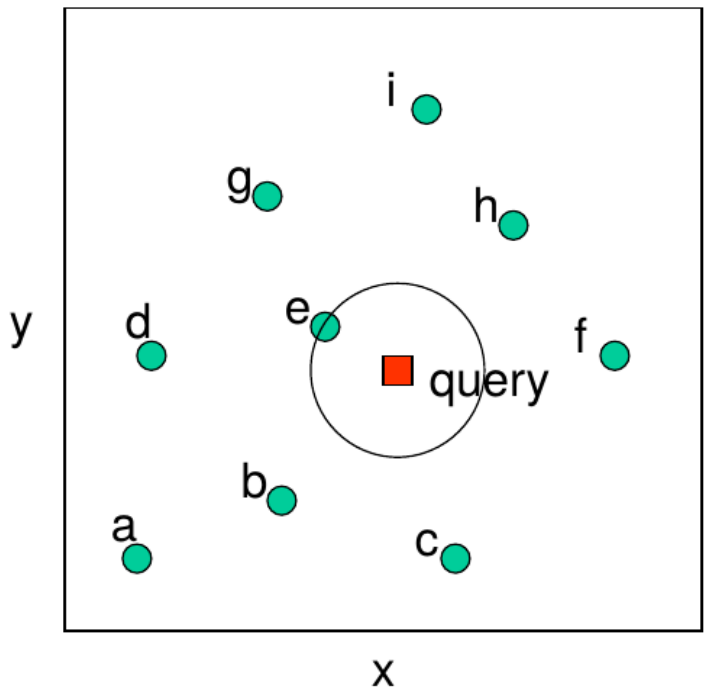


Rectangular query



Circular query

Nearest Neighbor Search



Nearest neighbor is e.

Outline

We have the following procedure

Steps

1. If there is just one point, form a leaf with that point.
2. Otherwise, divide the points in half by a line perpendicular to one of the axes.
3. Recursively construct k-d trees for the two sets of points.

We have the following procedure

Steps

1. If there is just one point, form a leaf with that point.
2. Otherwise, divide the points in half by a line perpendicular to one of the axes.
3. Recursively construct k-d trees for the two sets of points.

Question?

How do we divide the points?

We have the following procedure

Steps

1. If there is just one point, form a leaf with that point.
2. Otherwise, divide the points in half by a line perpendicular to one of the axes.
3. Recursively construct k-d trees for the two sets of points.

Question?

How do we divide the points?

We have the following procedure

Steps

1. If there is just one point, form a leaf with that point.
2. Otherwise, divide the points in half by a line perpendicular to one of the axes.
3. Recursively construct k-d trees for the two sets of points.

Question?

How do we divide the points?

Division Strategies

Criterion I

Divide points perpendicular to the axis with widest spread.

Criterion II

Divide in a round-robin fashion (book does it this way).

Division Strategies

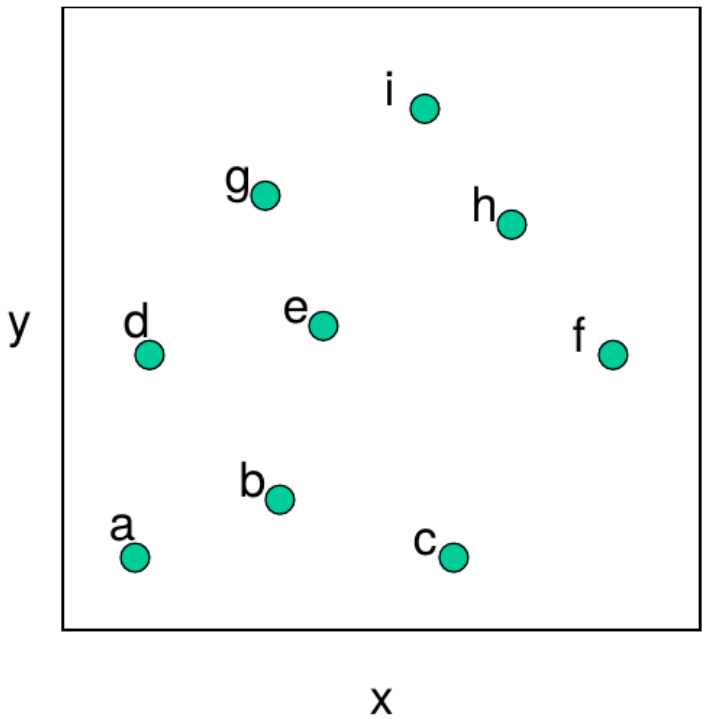
Criterion I

Divide points perpendicular to the axis with widest spread.

Criterion II

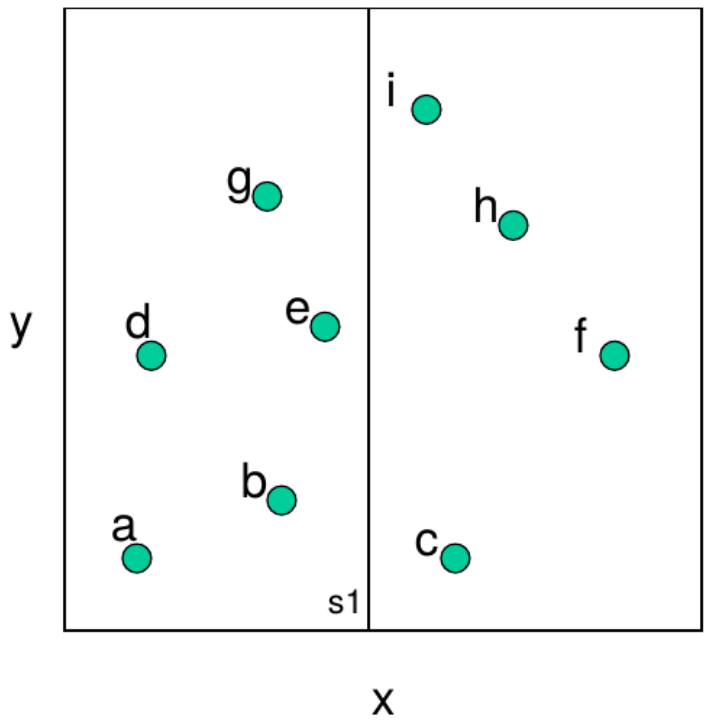
Divide in a round-robin fashion (book does it this way).

Example



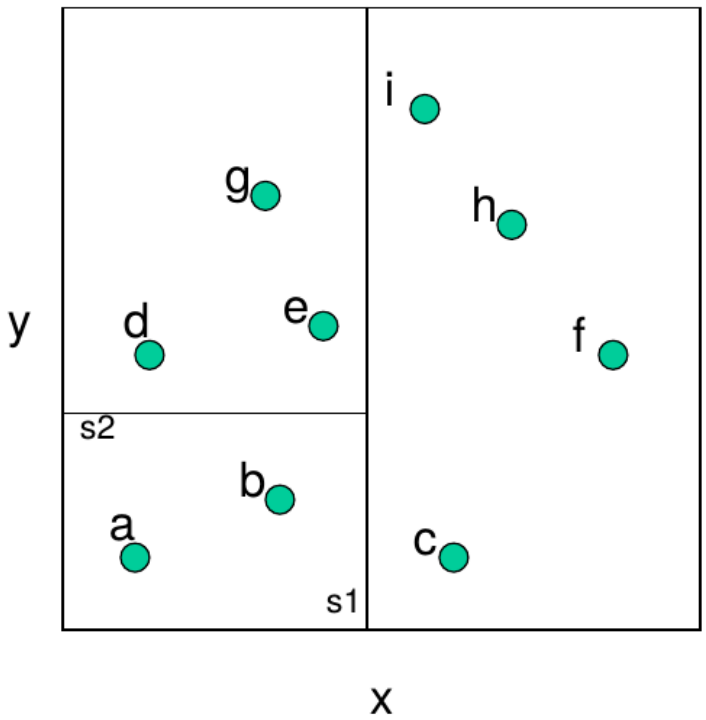
divide perpendicular to the widest spread.

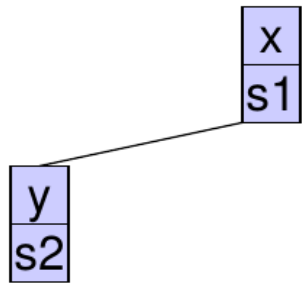
Example



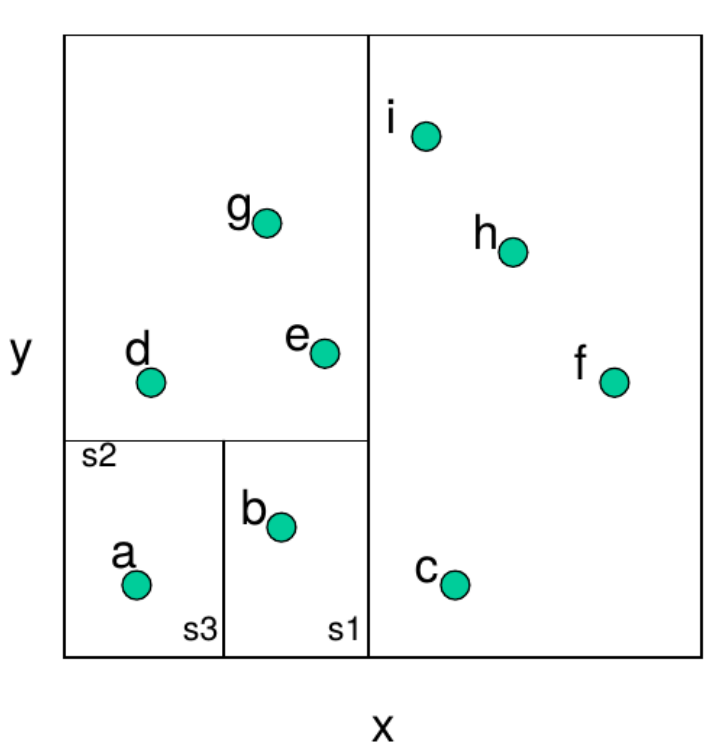
x
s1

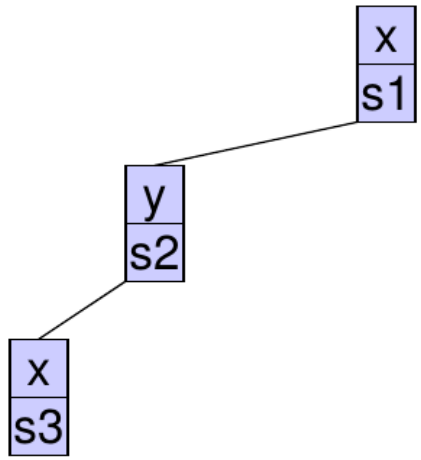
Example



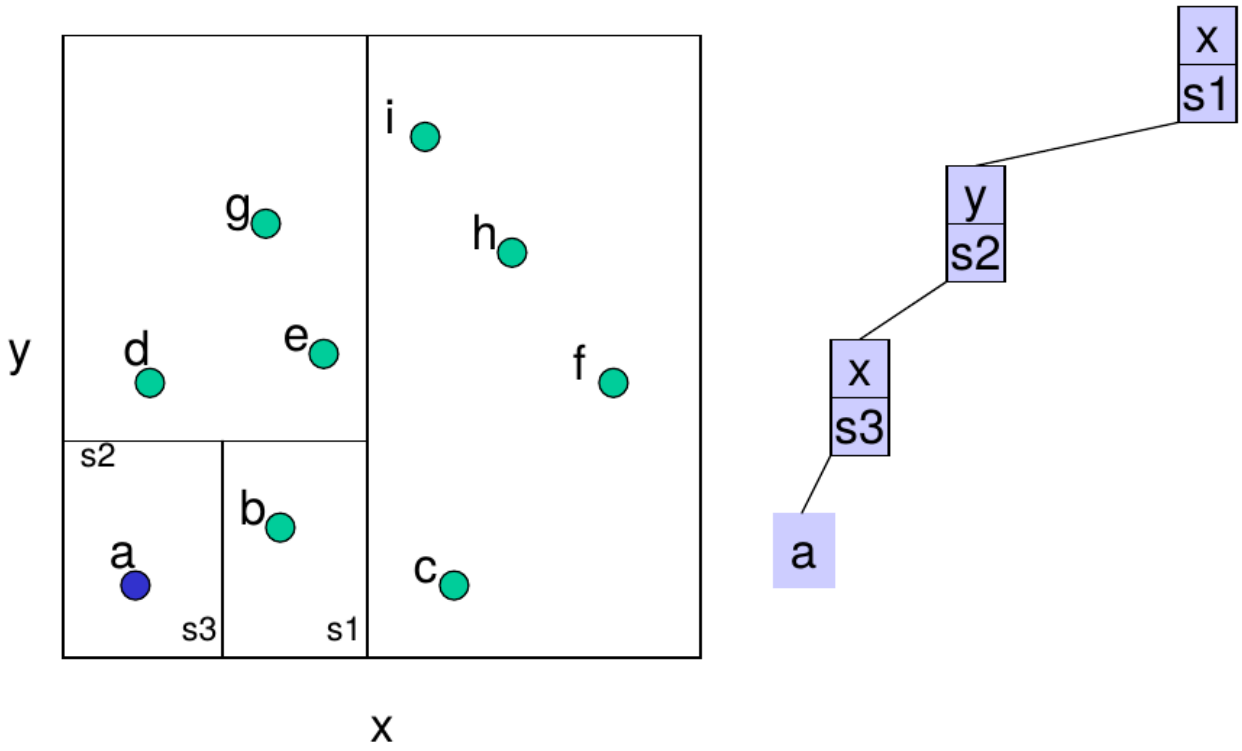


Example

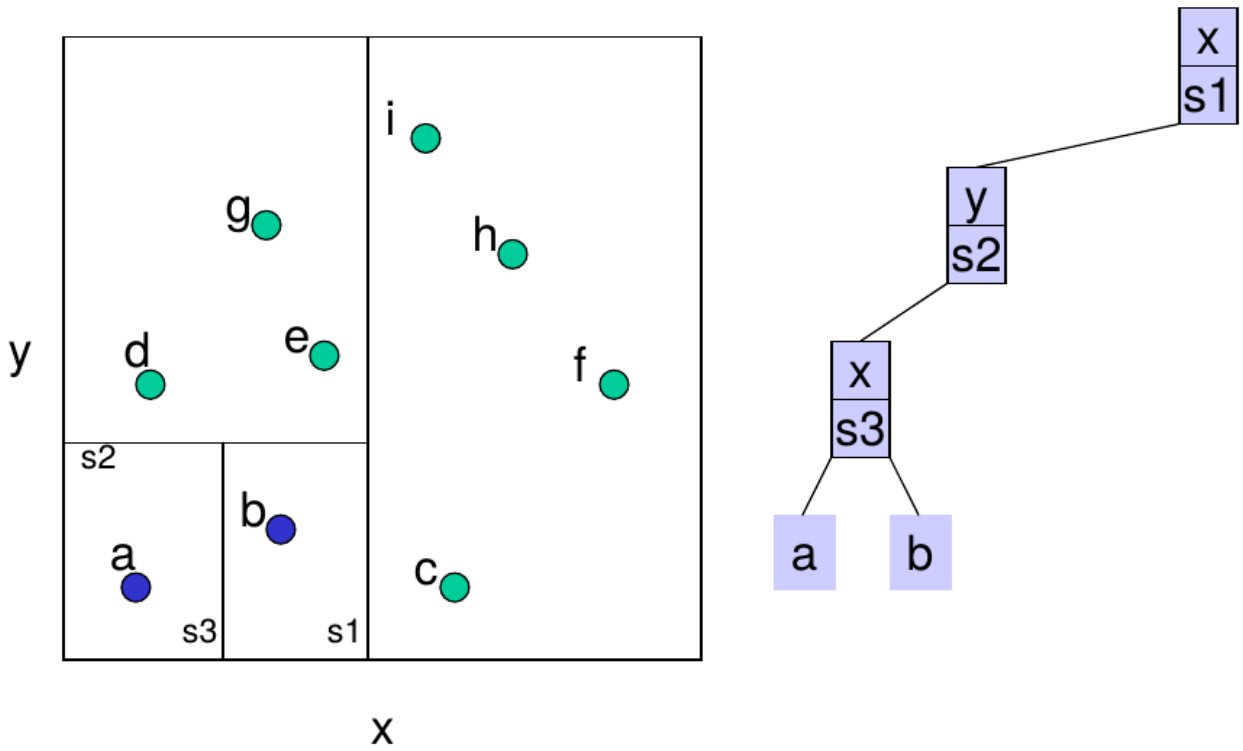




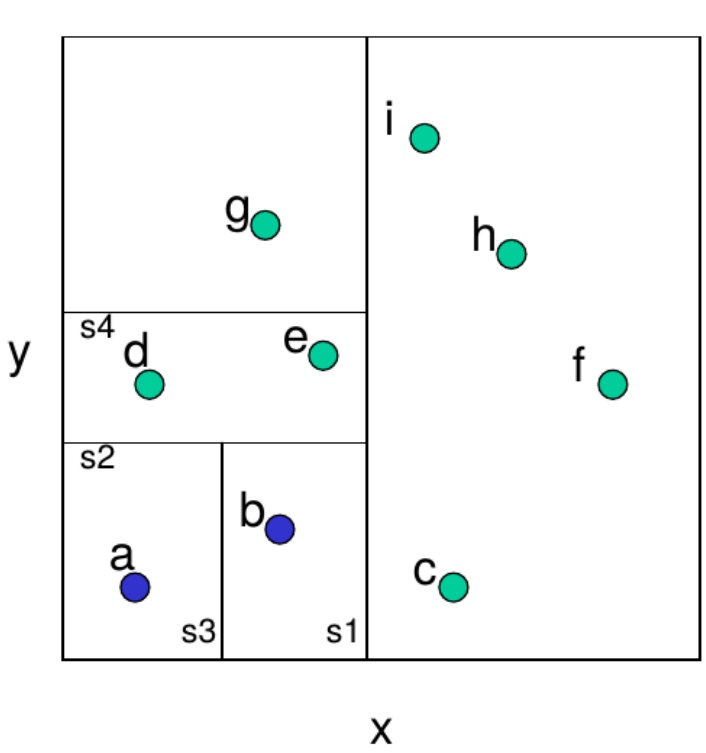
Example

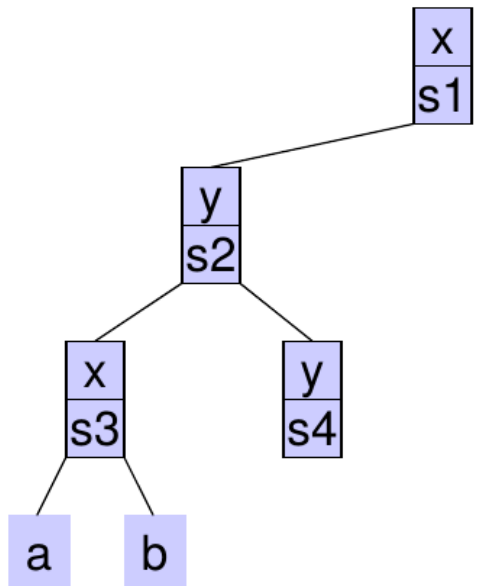


Example

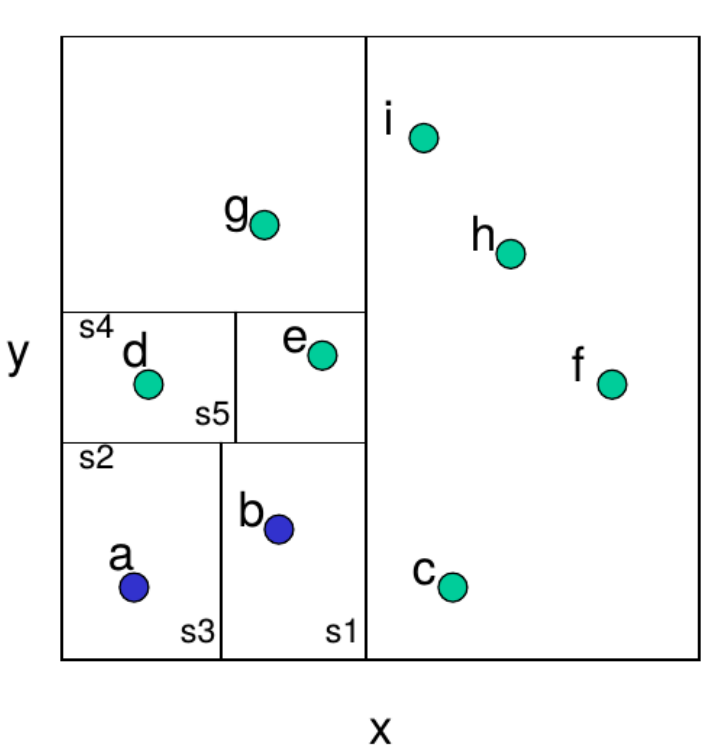


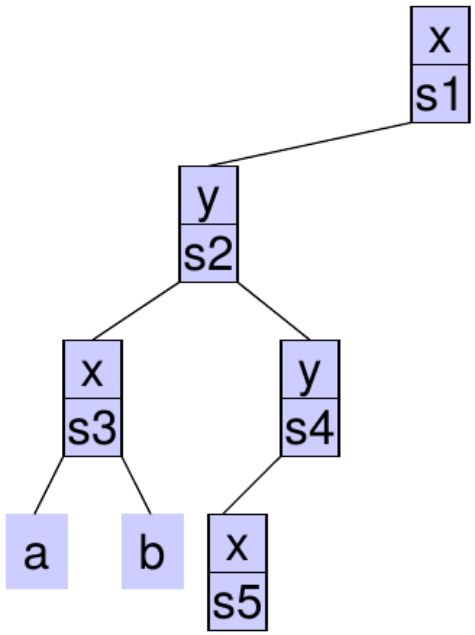
Example



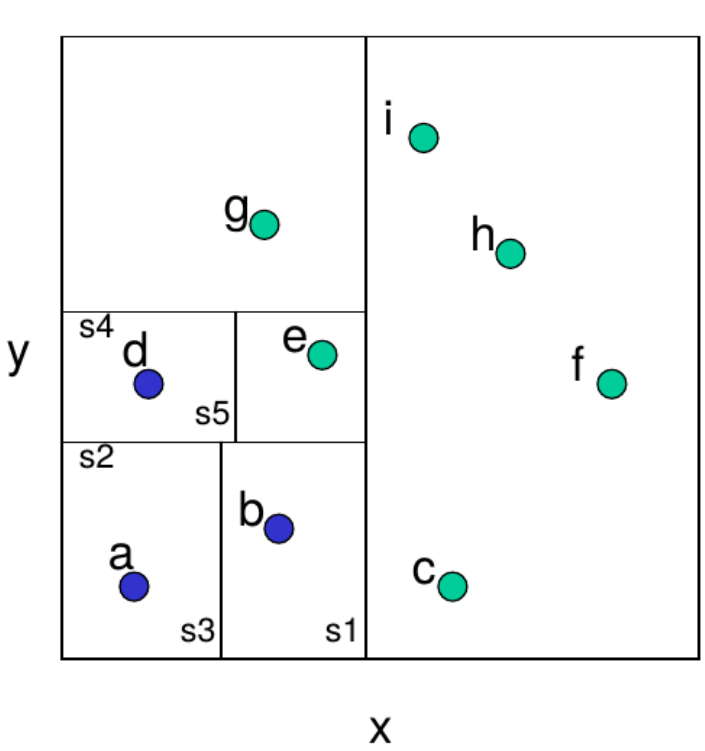


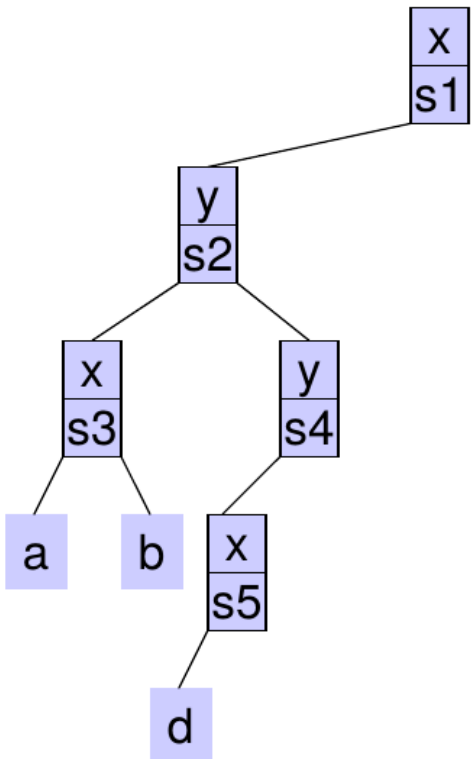
Example



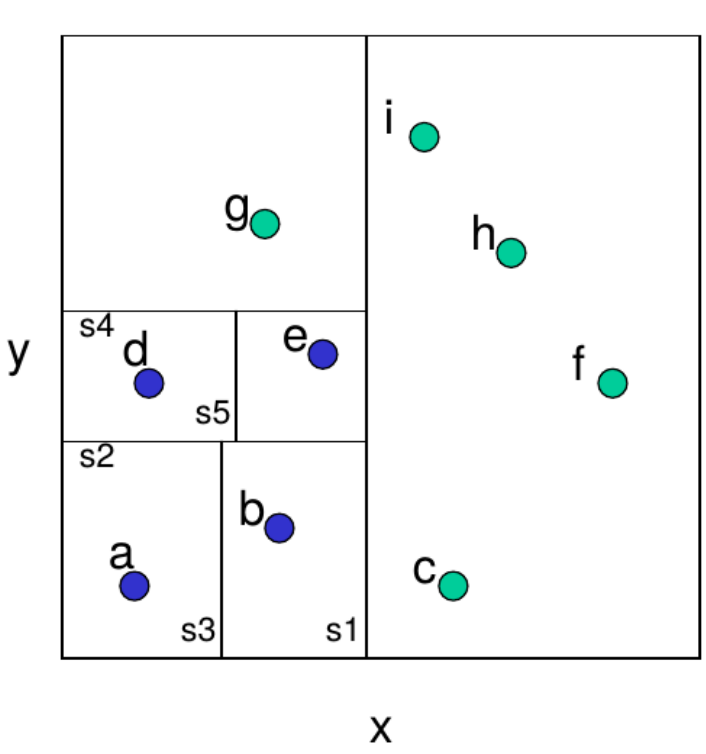


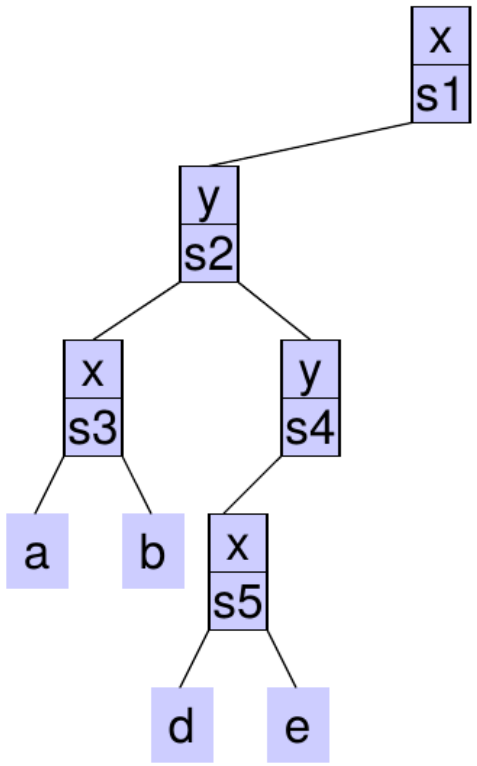
Example



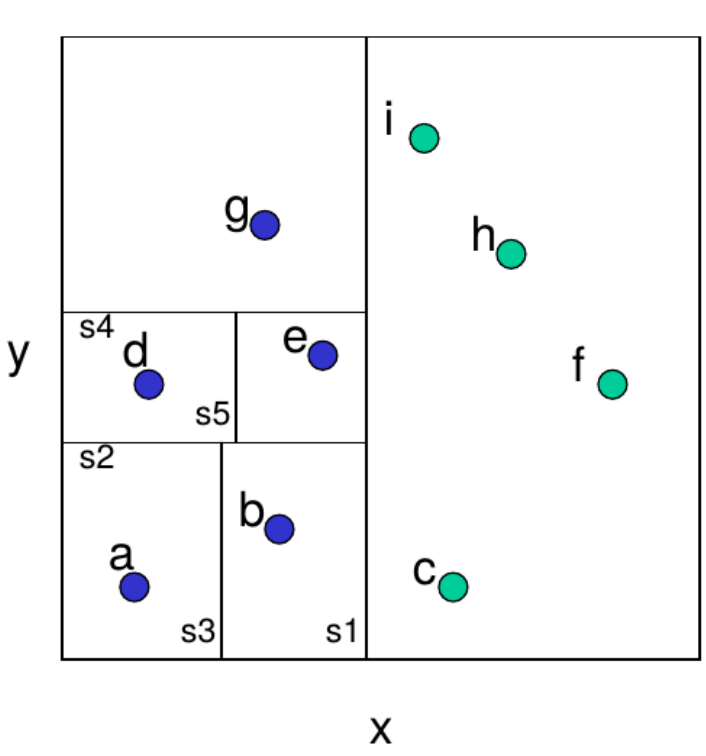


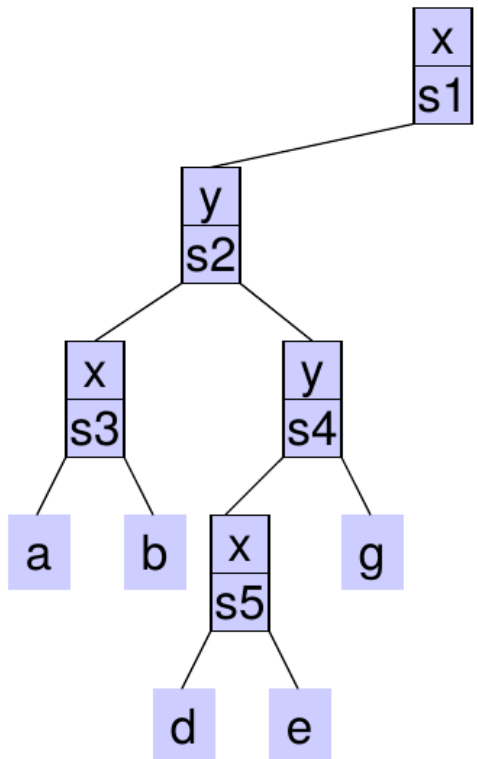
Example



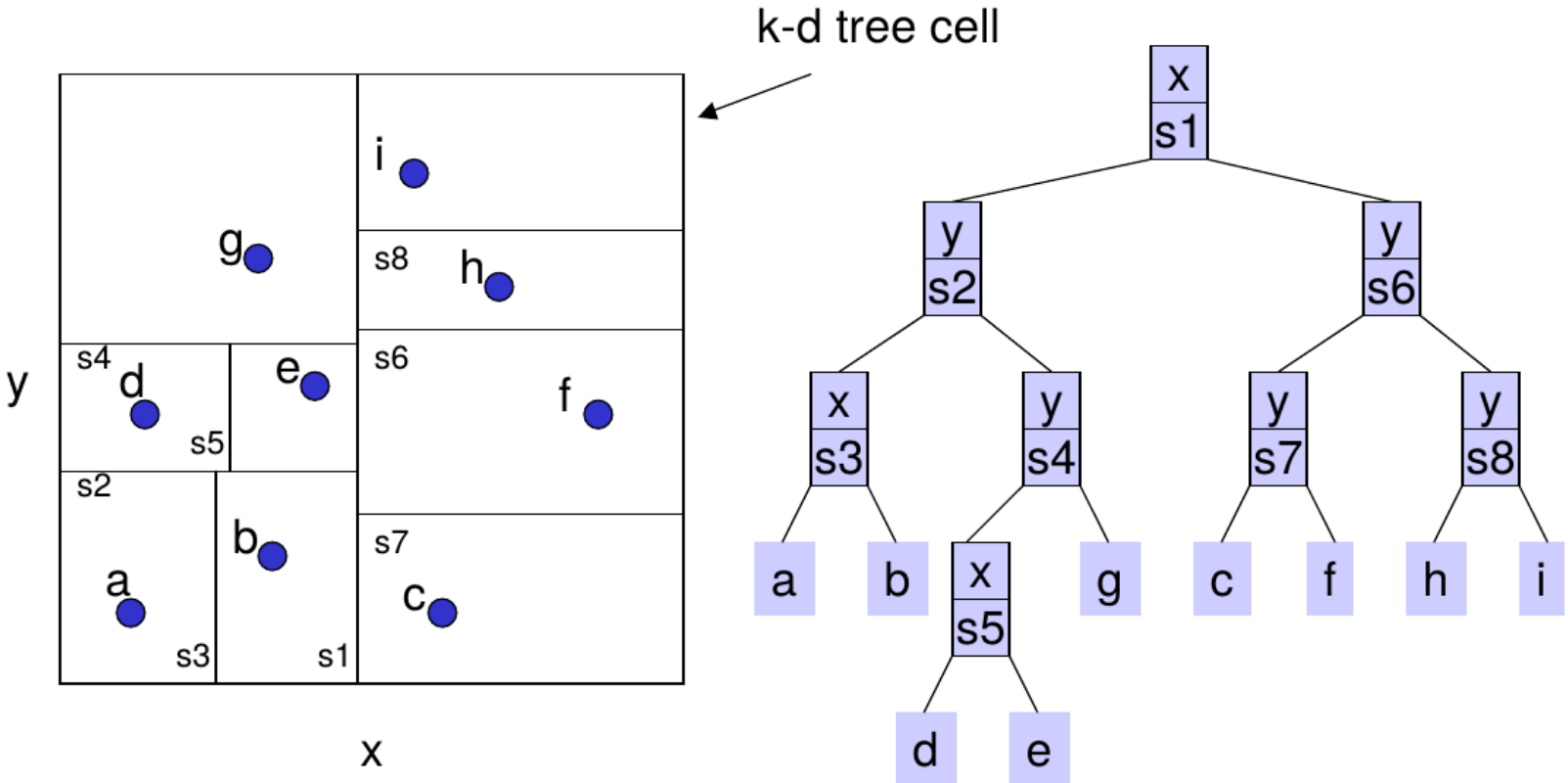


Example





Finally

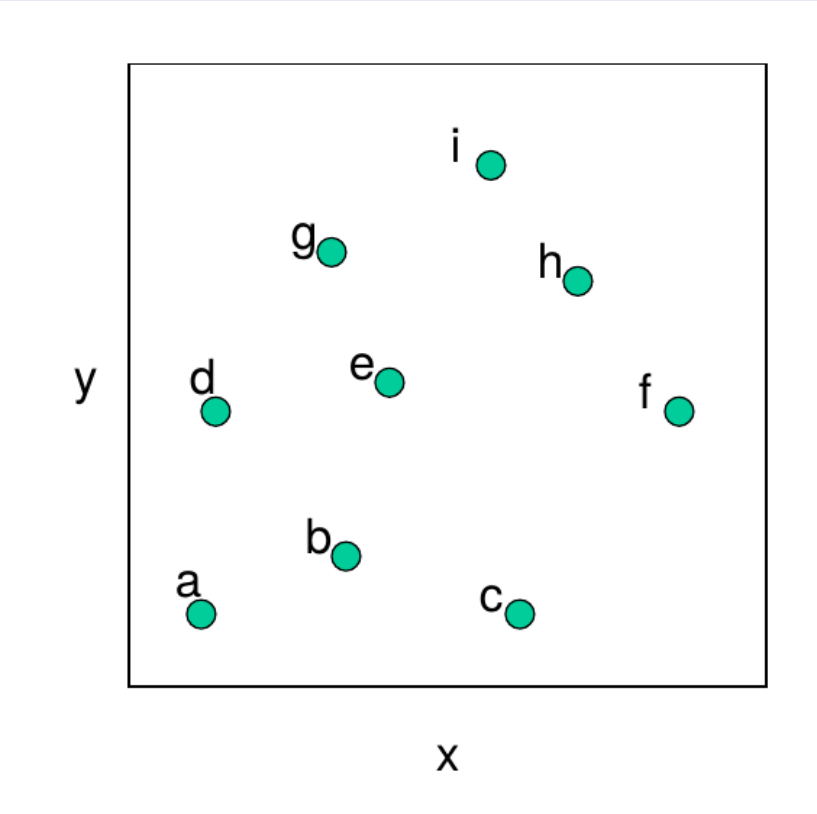

k-d tree cell
i
g
s8
h
s4
d
e
s6
f
y
s5
s2
b
s7
a
c
s3
s1
x
x
s1
y
s2
y
s6
x
s3
y
s4
y
s7
y
s8
a
b
x
s5
g
c
f
h
i
d
e

Outline

We have the following

In each dimension we have certain sorting

	1	2	3	4	5	6	7	8	9
x	a	d	g	b	e	i	c	h	f
y	a	c	b	d	f	e	h	g	i



Therefore

The max spread is the argument that maximize the following quantities

$$\arg\max f_x - a_x$$
$$\arg\max g_y - a_y$$

Basically

- In the selected dimension the middle point in the list splits the data.

Therefore

- To build the sorted lists for the other dimensions scan the sorted list adding each point to one of two sorted lists.

Therefore

The max spread is the argument that maximize the following quantities

$$\arg\max f_x - a_x$$
$$\arg\max g_y - a_y$$

Basically

- In the selected dimension the middle point in the list splits the data.

Therefore

- To build the sorted lists for the other dimensions scan the sorted list adding each point to one of two sorted lists.

Therefore

The max spread is the argument that maximize the following quantities

$$\arg\max f_x - a_x$$
$$\arg\max g_y - a_y$$

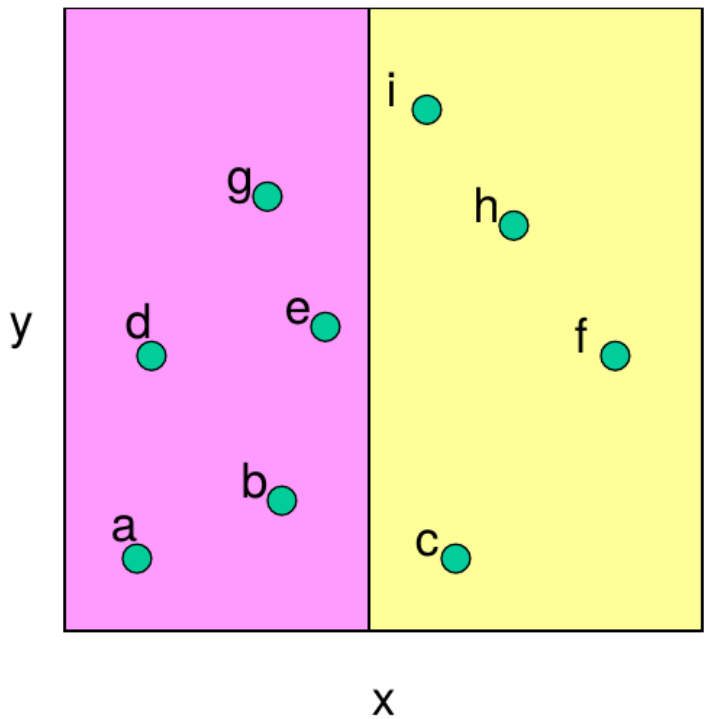
Basically

- In the selected dimension the middle point in the list splits the data.

Therefore

- To build the sorted lists for the other dimensions scan the sorted list adding each point to one of two sorted lists.

Then



sorted points in each dimension

	1	2	3	4	5	6	7	8	9
x	a	d	g	b	e	i	c	h	f
y	a	c	b	d	f	e	h	g	i

indicator for each set

a	b	c	d	e	f	g	h	i
0	0	1	0	0	1	0	1	1

scan sorted points in y dimension and add to correct set

y	a	b	d	e	g	c	f	h	i

Outline

KD Tree Construction Complexity

First sort the points in each dimension

- $O(dn \log n)$ time and dn storage.
- These are stored in matrices $A[1..d, 1..n]$

Finding the widest spread and equally divide

Into two subsets can be done in O(dn) time.

We have the recurrence

$$T(n, d) = 2T\left(\frac{n}{2}, d\right) + O(dn)$$

KD Tree Construction Complexity

First sort the points in each dimension

- $O(dn \log n)$ time and dn storage.
- These are stored in matrices $A[1..d, 1..n]$

Finding the widest spread and equally divide

Into two subsets can be done in O(dn) time.

We have the recurrence

$$T(n, d) = 2T\left(\frac{n}{2}, d\right) + O(dn)$$

KD Tree Construction Complexity

First sort the points in each dimension

- $O(dn \log n)$ time and dn storage.
- These are stored in matrices $A[1..d, 1..n]$

Finding the widest spread and equally divide

Into two subsets can be done in O(dn) time.

We have the recurrence

$$T(n, d) = 2T\left(\frac{n}{2}, d\right) + O(dn)$$

Therefore

Constructing the KD Tree can be done in

- Time $O(n \log n)$
- Space $O(n)$

Outline

Node Structure

A node has 5 fields

- Axis (splitting axis)
- Value (splitting value)
- left (left subtree)
- right (right subtree)
- point (holds a point if left and right children are null)

Outline

Complexity

Theorem

A set of n points in the plane can be preprocessed in $O(dn \log n)$ time into a data structure of $O(dn)$ size so that any d-range query can be answered in time $O(d\sqrt{n} + k)$, where k is the number of answers reported

Outline

Encoding Sets as Bit Vectors

Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encoding Sets as Bit Vectors

Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise AND, and set union as bitwise OR.

Encoding Sets as Bit Vectors

Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**.

Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 1/4$

Encoding Sets as Bit Vectors

Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR.**

Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 1/4$

Encoding Sets as Bit Vectors

Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR.**

Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 1/4$

Encoding Sets as Bit Vectors

Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR.**

Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1-(\text{Jaccard similarity}) = 1/4$

Encoding Sets as Bit Vectors

Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR.**

Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 1/4$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - Typical matrix is sparse!

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 3/6$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - Typical matrix is sparse!

Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 3/6$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 3/6$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

Each document is a column

- Example: $sim(C_1, C_2) =$?
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 3/6$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

Each document is a column

- Example: $sim(C_1, C_2) =$?
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 3/6$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

Each document is a column

- Example: $sim(C_1, C_2) =$?
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1$–(Jaccard similarity) $= 3/6$

Outline

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)

- These methods can produce false negatives, and even false positives (if the optional check is not made)

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Finally, The Buckets

Buckets

- Thus, we hash documents into buckets, and expect that "most" pairs of near duplicate docs hash into the same bucket!

Outline

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: Min-hashing.

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: Min-hashing.

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: Min-hashing.

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: **Min-hashing.**

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: **Min-hashing.**

Min-Hashing

Random permutation

Imagine the rows of the boolean matrix permuted under random permutation π .

"Hash" function $h_\pi(C)$

- Define a "hash" function $h_\pi(C) =$ the number of the first (in the permuted order π) row in which column C has value 1:

$$h_\pi(C) = min_\pi \pi(C)$$

What can we do?

- Use several (e.g., 100) independent hash functions to create a signature of a column

Min-Hashing

Random permutation

Imagine the rows of the boolean matrix permuted under random permutation π .

"Hash" function $h_\pi(C)$

- Define a "hash" function $h_\pi(C) =$ the number of the first (in the permuted order π) row in which column C has value 1:

$$h_\pi(C) = min_\pi \pi(C)$$

What can we do?

- Use several (e.g., 100) independent hash functions to create a signature of a column

Min-Hashing

Random permutation

Imagine the rows of the boolean matrix permuted under random permutation π .

"Hash" function $h_\pi(C)$

- Define a "hash" function $h_\pi(C) =$ the number of the first (in the permuted order π) row in which column C has value 1:

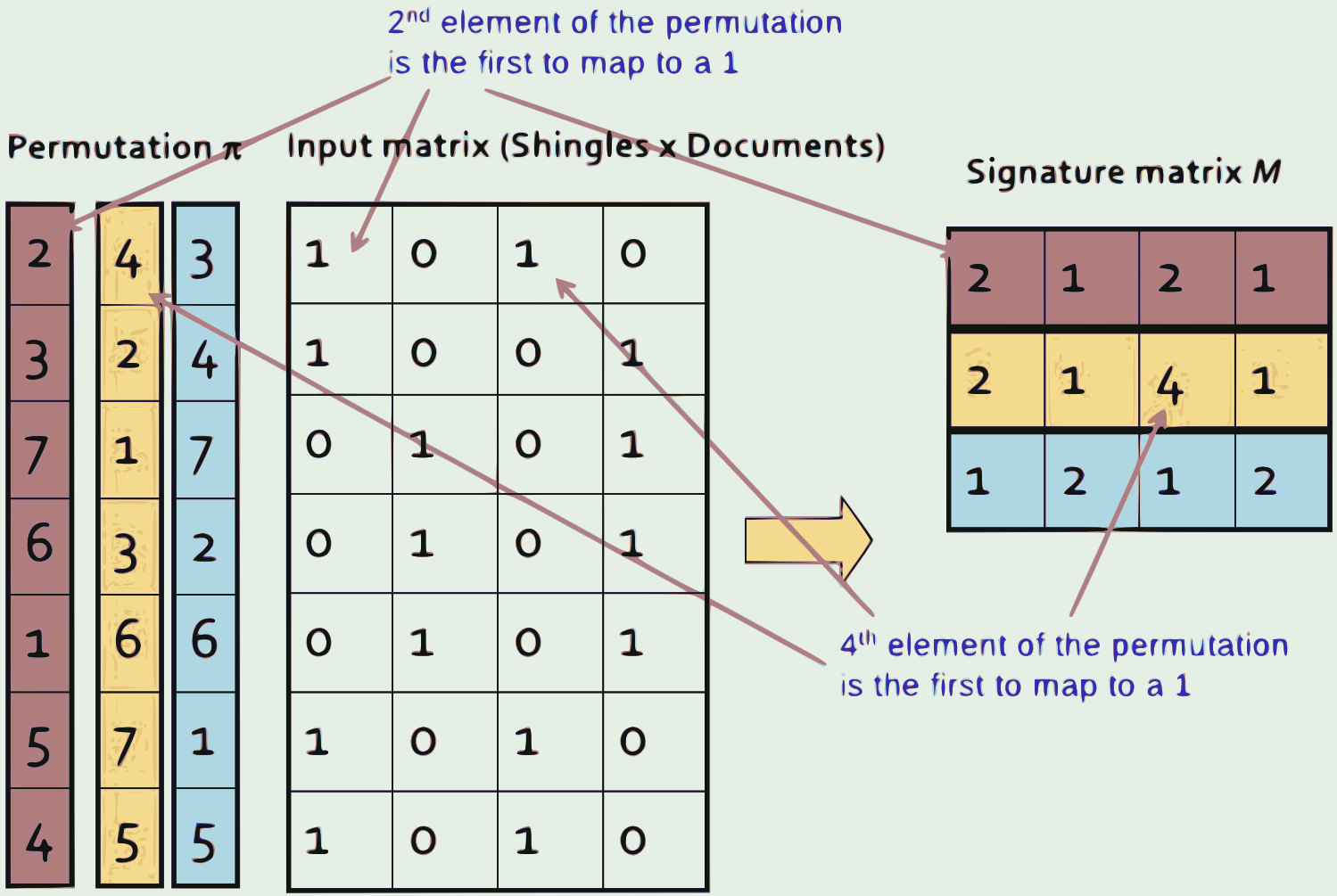
$$h_\pi(C) = min_\pi \pi(C)$$

What can we do?

- Use several (e.g., 100) independent hash functions to create a signature of a column

Min-Hashing Example

Something Notable



Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (1)$$

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (1)$$

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (1)$$

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (1)$$

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (1)$$

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (1)$$

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (1)$$

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (1)$$

Surprising Property

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2) \quad (1)$$

Four Types of Rows between two Documents

Given cols C_1 and C_2, rows may be classified as

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

$a = \#$rows of type A, etc.

Note

$$sim(C_1, C_2) = \frac{a}{a+b+c} \tag{2}$$

Then

- Then: $Pr[h(C_1) = h(C_2)] = sim(C_1, C_2)$
  - Look down the cols C_1 and C_2 until we see a 1
  - If it's a type-A row then $h(C_1) = h(C_2)$ If a type-B or type-C row then not

Four Types of Rows between two Documents

Given cols C_1 and C_2, rows may be classified as

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

$a = \#$rows of type A, etc.

Note

$$sim(C_1, C_2) = \frac{a}{a + b + c} \tag{2}$$

Then

- Then: $Pr[h(C_1) = h(C_2)] = sim(C_1, C_2)$
  - Look down the cols C_1 and C_2 until we see a 1.
  - If it's a type-A row then $h(C_1) = h(C_2)$... if it's a type-B or type-C row then not.

Four Types of Rows between two Documents

Given cols C_1 and C_2, rows may be classified as

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

$a = \#$rows of type A, etc.

Note

$$sim(C_1, C_2) = \frac{a}{a+b+c} \tag{2}$$

Then

- Then: $Pr[h(C_1) = h(C_2)] = sim(C_1, C_2)$
  - Look down the cols C_1 and C_2 until we see a 1.
  - If it's a type-A row, then $h(C_1) = h(C_2)$ If a type-B or type-C row, then not.

Four Types of Rows between two Documents

Given cols C_1 and C_2, rows may be classified as

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

$a = \#$rows of type A, etc.

Note

$$sim(C_1, C_2) = \frac{a}{a+b+c} \tag{2}$$

Then

- Then: $Pr[h(C_1) = h(C_2)] = sim(C_1, C_2)$
  - Look down the cols C_1 and C_2 until we see a 1.
  - If it's a type-A row, then $h(C_1) = h(C_2)$ If a type-B or type-C row, then not.

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity

- The similarity of two signatures is the fraction of the hash functions in which they agree

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity

- The similarity of two signatures is the fraction of the hash functions in which they agree

Note

- Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity

- The similarity of two signatures is the fraction of the hash functions in which they agree

Note

- Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity

- The similarity of two signatures is the fraction of the hash functions in which they agree

Note

- Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures

Min-Hashing Example

Example

Permutation π | Input matrix (Shingles x Documents)

Permutation π			Input matrix (Shingles x Documents)			
2	4	3	1	0	1	0
3	2	4	1	0	0	1
7	1	7	0	1	0	1
6	3	2	0	1	0	1
1	6	6	0	1	0	1
5	7	1	1	0	1	0
4	5	5	1	0	1	0

Signature matrix M

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i] =$according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i] =$according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

- We achieved our goal! We "compressed" long bit vectors into short signatures

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i]$ =according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

- We achieved our goal! We "compressed" long bit vectors into short signatures

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i]$ =according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

- We achieved our goal! We "compressed" long bit vectors into short signatures

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i]$ =according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

- We achieved our goal! We "compressed" long bit vectors into short signatures

Outline

Implementation Trick

- Permuting rows even once is prohibitive

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?

Universal hashing:

$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$

where:

a, b... random integers

p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:

$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \text{ mod } p) \text{ mod } N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \text{ mod } p) \text{ mod } N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \text{ mod } p) \text{ mod } N$
where:
a, b... random integers
p... prime number $(p > N)$

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \text{ mod } p) \text{ mod } N$
where:
a, b... random integers
p... prime number $(p > N)$

Outline

Trying to define LSH

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

Trying to define LSH

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

LSH – General idea

- Use a function $f(x, y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated.

Trying to define LSH

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

LSH – General idea

- Use a function $f(x, y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated.

For MinHash matrices

- Hash columns of signature matrix M to many buckets.
- Each pair of documents that hashes into the same bucket is a candidate pair.

Trying to define LSH

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

LSH – General idea

- Use a function $f(x, y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated.

For MinHash matrices

- Hash columns of signature matrix M to many buckets.
- Each pair of documents that hashes into the same bucket is a candidate pair.

Trying to define LSH

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

LSH – General idea

- Use a function $f(x, y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated.

For MinHash matrices

- Hash columns of signature matrix M to many buckets.
- Each pair of documents that hashes into the same bucket is a candidate pair.

Candidates from Minhash

- Pick a similarity threshold s $(0 < s < 1)$.

Candidates from Minhash

- Pick a similarity threshold s $(0 < s < 1)$.

Candidate pair

- Columns x and y of M are a candidate pair if their signatures agree on at least fraction s of their rows:
  - $M(i, x) = M(i, y)$ for at least fraction s of values of i
    - We expect documents x and y to have the same (Jaccard) similarity as is the similarity of their signatures

Candidates from Minhash

- Pick a similarity threshold s $(0 < s < 1)$.

Candidate pair

- Columns x and y of M are a candidate pair if their signatures agree on at least fraction s of their rows:
  - $M(i, x) = M(i, y)$ for at least fraction s of values of i
    - We expect documents x and y to have the same (Jaccard) similarity as is the similarity of their signatures

Candidates from Minhash

- Pick a similarity threshold s $(0 < s < 1)$.

Candidate pair

- Columns x and y of M are a candidate pair if their signatures agree on at least fraction s of their rows:
  - $M(i, x) = M(i, y)$ for at least fraction s of values of i
    - ★ We expect documents x and y to have the same (Jaccard) similarity as is the similarity of their signatures

LSH for Minhash

Big idea

- Hash columns of signature matrix M several times

Likely to hash

- Arrange that (only) similar columns are likely to hash to the same bucket, with high probability

Candidate pairs

- Candidate pairs are those that hash to the same bucket

LSH for Minhash

Big idea

- Hash columns of signature matrix M several times

Likely to hash

- Arrange that (only) similar columns are likely to hash to the same bucket, with high probability

Candidate pairs

- Candidate pairs are those that hash to the same bucket

LSH for Minhash

Big idea

- Hash columns of signature matrix M several times

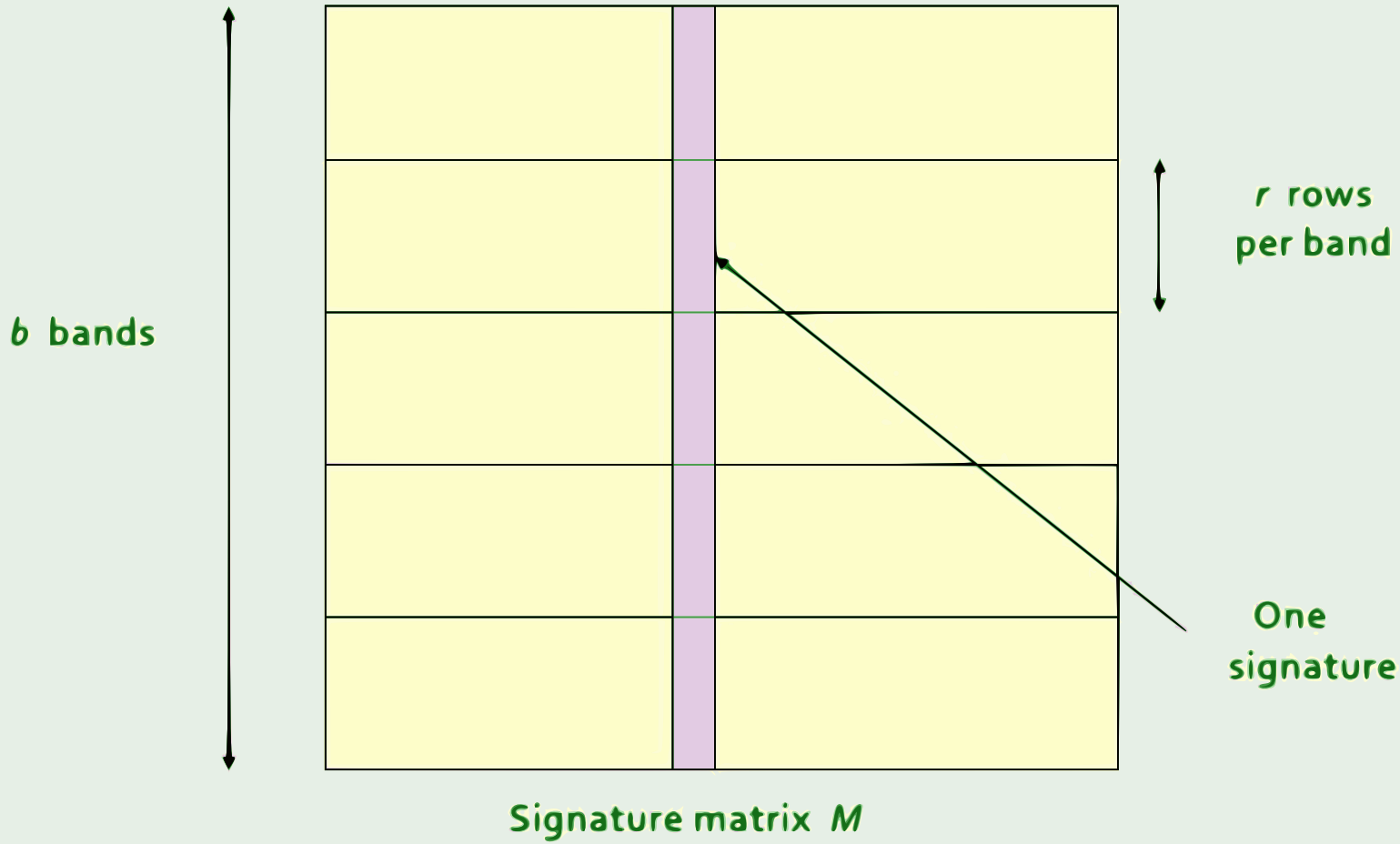
Likely to hash

- Arrange that (only) similar columns are likely to hash to the same bucket, with high probability

Candidate pairs

- Candidate pairs are those that hash to the same bucket

Partition M into b Bands



Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune b and r to catch most similar pairs, but few non-similar pairs.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune b and r to catch most similar pairs, but few non-similar pairs.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune b and r to catch most similar pairs, but few non-similar pairs.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

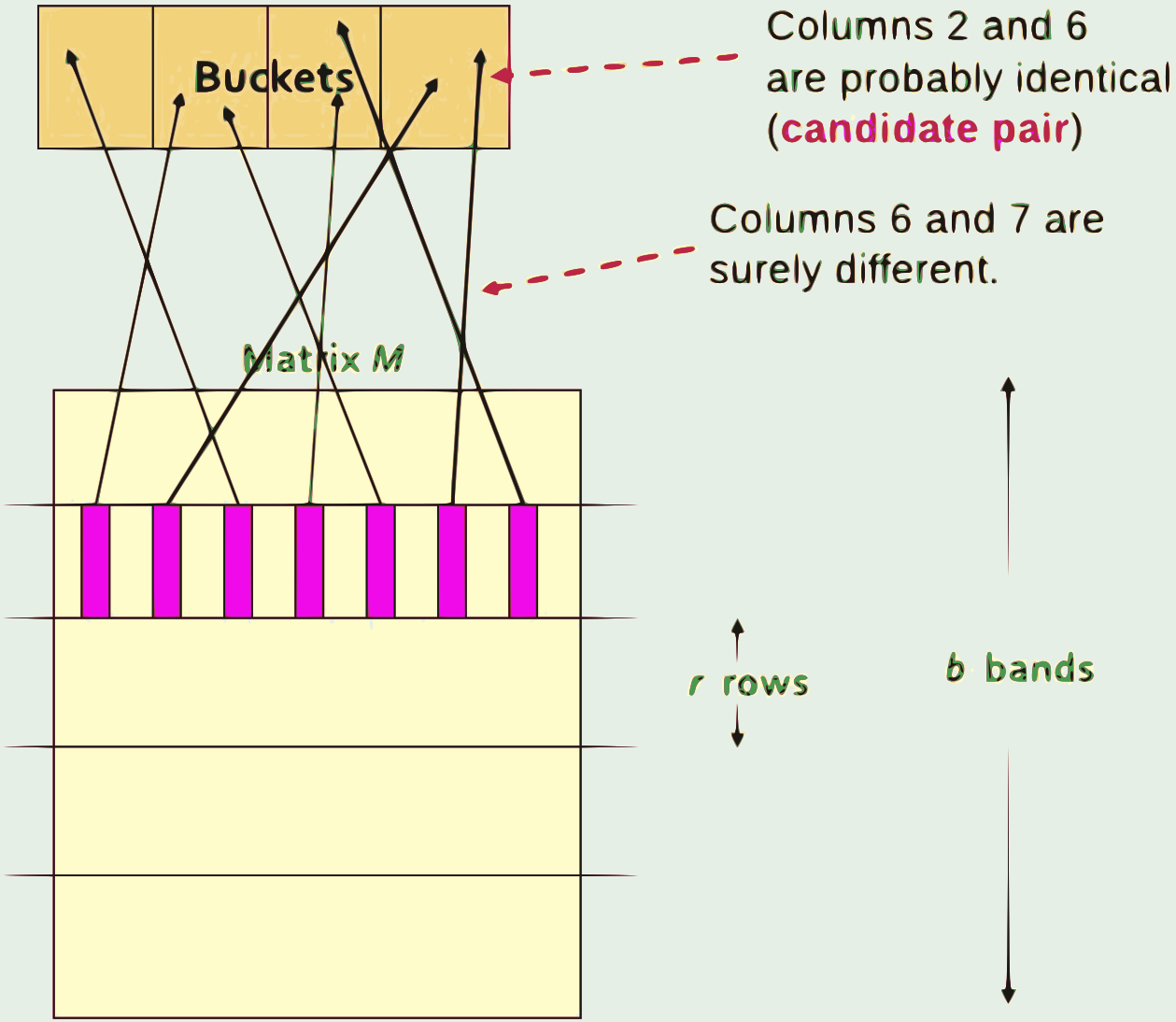
Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune b and r to catch most similar pairs, but few non-similar pairs.

Hashing Bands



Simplifying Assumption

Identical

- There are enough buckets that columns are unlikely to hash to the same bucket unless they are identical in a particular band

Same bucket

- Then, we assume that "same bucket" means "identical in that band"

Not for correctness

- Assumption needed only to simplify analysis, not for the correctness of algorithm

Simplifying Assumption

Identical

- There are enough buckets that columns are unlikely to hash to the same bucket unless they are identical in a particular band

Same bucket

- Then, we assume that "same bucket" means "identical in that band"

Not for correctness

- Assumption needed only to simplify analysis, not for the correctness of algorithm

Simplifying Assumption

Identical

- There are enough buckets that columns are unlikely to hash to the same bucket unless they are identical in a particular band

Same bucket

- Then, we assume that "same bucket" means "identical in that band"

Not for correctness

- Assumption needed only to simplify analysis, not for the correctness of algorithm

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Goal

- Find pairs of documents that are at least $s = 0.8$ similar

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Goal

- Find pairs of documents that are at least $s = 0.8$ similar

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Goal

- Find pairs of documents that are at least $s = 0.8$ similar

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Goal

- Find pairs of documents that are at least $s = 0.8$ similar

Now, if C_1,C_2 are 80% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

Now, if C_1,C_2 are 80% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

Now, if C_1,C_2 are 80% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about $1/3000$th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about $1/3000$th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about $1/3000$th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about $1/3000$th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

C_1,C_2 are 30% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

C_1,C_2 are 30% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$ we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

C_1,C_2 are 30% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

C_1,C_2 are 30% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

C_1,C_2 are 30% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

C_1,C_2 are 30% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

C_1,C_2 are 30% Similar

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - ⋆ They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

LSH Involves a Tradeoff

You need to pick

- The number of minhashes (rows of M).
- The number of bands b.
- The number of rows r per band to balance false positives/negatives.

LSH Involves a Tradeoff

You need to pick

- The number of minhashes (rows of M).
- The number of bands b.
- The number of rows r per band to balance false positives/negatives.

Example

- if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

LSH Involves a Tradeoff

You need to pick

- The number of minhashes (rows of M).
- The number of bands b.
- The number of rows r per band to balance false positives/negatives.

Example

- if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

LSH Involves a Tradeoff

You need to pick

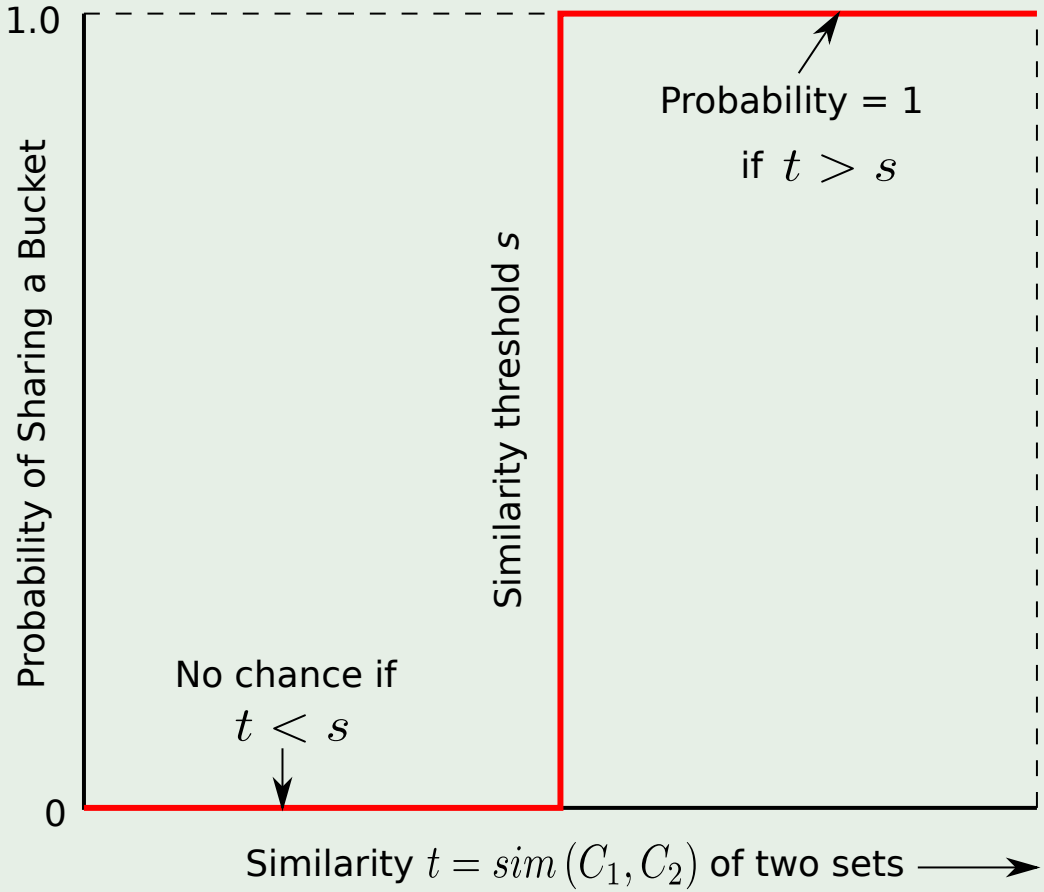
- The number of minhashes (rows of M).
- The number of bands b.
- The number of rows r per band to balance false positives/negatives.

Example

- if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

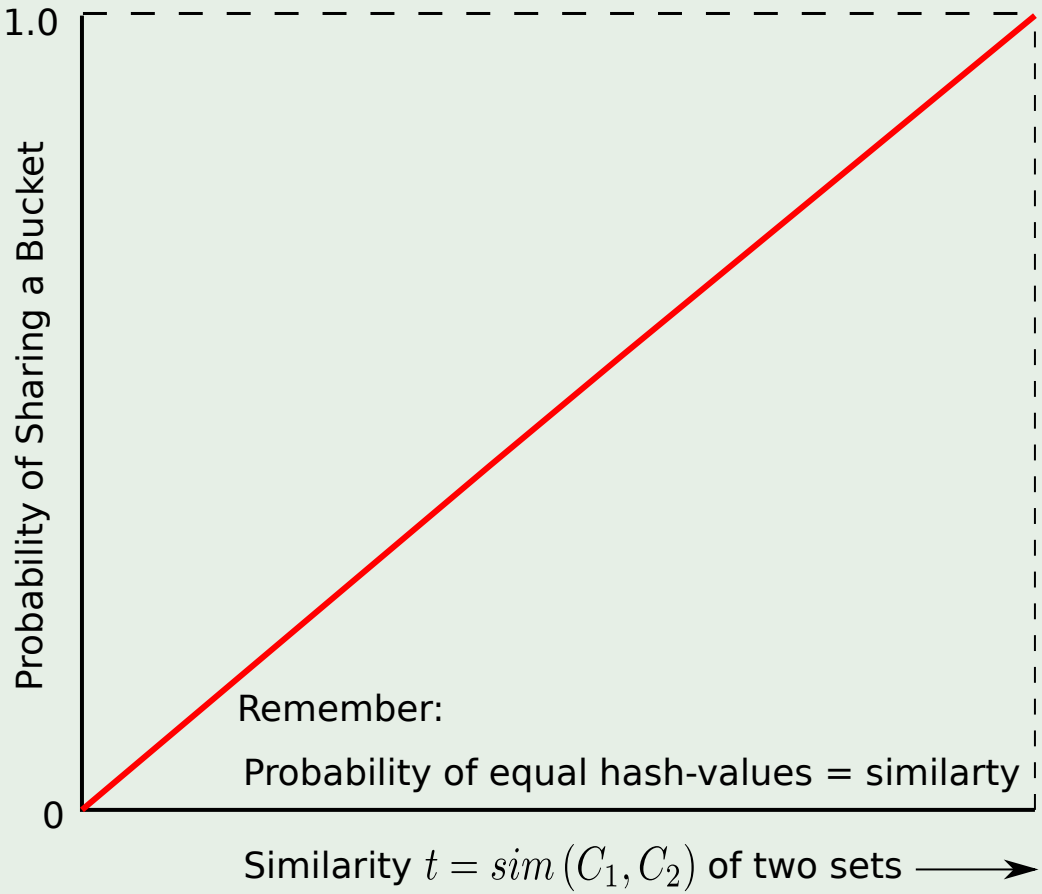
Analysis of LSH - What We Want

The Ideal detection of similar objects



What 1 Band of 1 Row Gives You

Not so great



Given that probability of two documents aggree in a row is s

We can calculate the probability that these documents become a candidate pair as follows

1. The probability that the signatures agree in all rows of one particular band is s^r.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

Given that probability of two documents aggree in a row is s

We can calculate the probability that these documents become a candidate pair as follows

1. The probability that the signatures agree in all rows of one particular band is s^r.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

Given that probability of two documents aggree in a row is s

We can calculate the probability that these documents become a candidate pair as follows

1. The probability that the signatures agree in all rows of one particular band is s^r.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

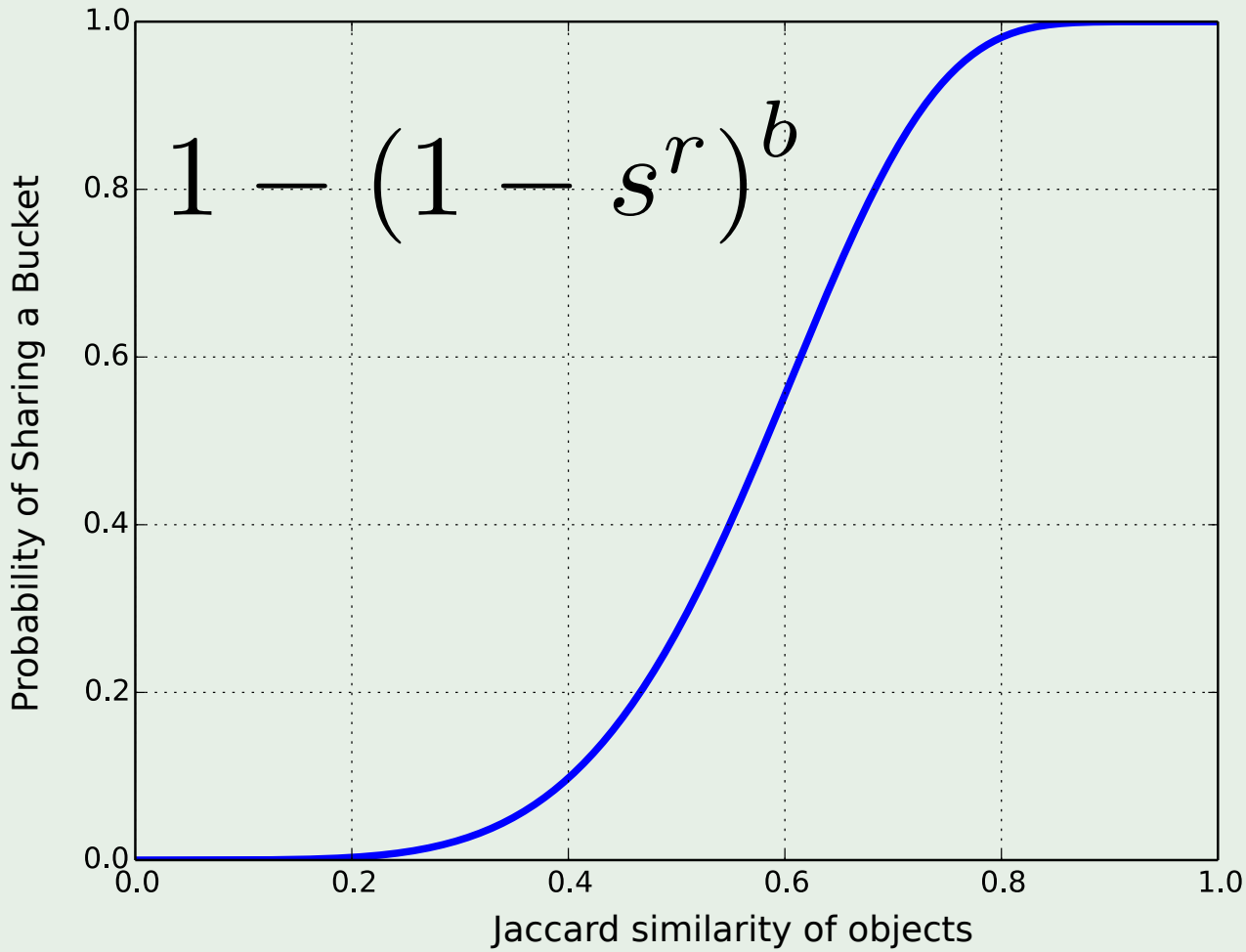
Given that probability of two documents aggree in a row is s

We can calculate the probability that these documents become a candidate pair as follows

1. The probability that the signatures agree in all rows of one particular band is s^r.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

If you fix r and b

Something Notable



Example: $b = 20$; $r = 5$

Given

- Similarity threshold s

Similarity threshold s Prob. that at least 1 band is identical

s	$1 - (1 - s^r)^b$
.2	0.006
.3	0.047
.4	0.186
.5	0.470
.6	0.802
.7	0.975
.8	0.9996

Example: $b = 20$; $r = 5$

Given

- Similarity threshold s

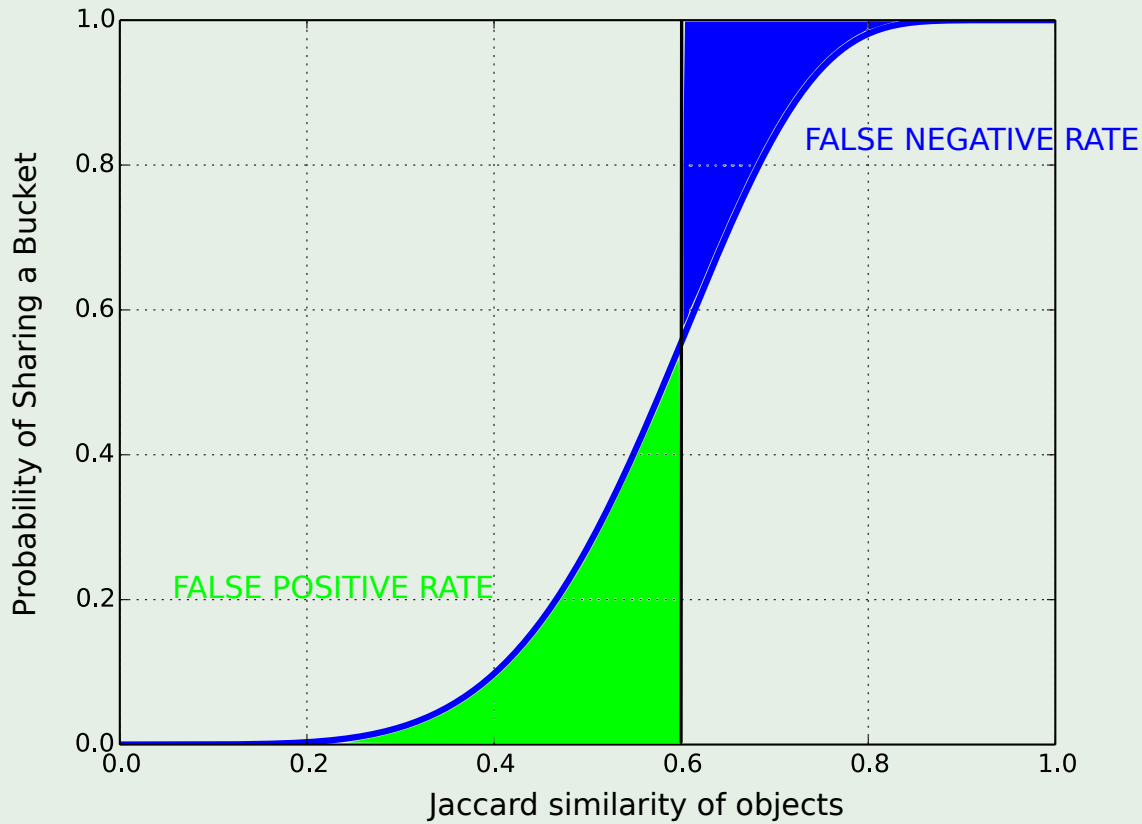
Similarity threshold s Prob. that at least 1 band is identical

s	$1 - (1 - s^r)^b$
.2	0.006
.3	0.047
.4	0.186
.5	0.470
.6	0.802
.7	0.975
.8	0.9996

Picking r and b: The S-curve

Picking r and b to get the best S-curve

- 50 hash-functions $(r = 5, b = 10)$



LSH Summary

Tune M, b, r

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures

Check in main memory

- Check in main memory that candidate pairs really do have similar signatures

Optional

- In another pass through data, check that the remaining candidate pairs really represent similar documents

LSH Summary

Tune M, b, r

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures

Check in main memory

- Check in main memory that candidate pairs really do have similar signatures

Optional

- In another pass through data, check that the remaining candidate pairs really represent similar documents

LSH Summary

Tune M, b, r

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures

Check in main memory

- Check in main memory that candidate pairs really do have similar signatures

Optional

- In another pass through data, check that the remaining candidate pairs really represent similar documents

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$